Optimisation

Master's degree in Modelling for Science and Engineering

Genetic algorithms.

Author:

Joseba Hernández Bravo

Teacher:

Lluís Alsedà

Barcelona, January 9, 2022

Contents

Chapter 1

Introduction

In this section we will briefly introduce the main ideas of the problem we are solving.

The main purpose of this work is to fit the free parameters $x_0, \varphi, \lambda, \mu, \sigma, \delta$ of equation 1.1 so that they fit as closely as possible to the actual data presented in table 1.

$$\frac{dx}{dt} = \varphi x - \beta x^2 - \lambda \psi(x, \mu, \sigma, \delta) \tag{1.1}$$

For this purpose, we will make use of the genetic algorithms with the following fitness function:

$$max\left\{ \Big(x(t) - z(t + 2016)\Big)^2 : t = 0, 1, 2, ..., 11 \right\} \tag{1.2}$$

To each parameter we have assigned a value within reasonable search range. Since in the rest of the program we will be working with the binary representations of these parameters, it is convenient to re-scale the values to unsigned int parameters. The values have been re-scaled in the following form:

- x_0: from $[0, 16600]$ to $[0, 2^{21} - 1]$
- φ: from $[-100, 0.35]$ to $[0, 2^{34} - 1]$
- λ: from $[0, 3000]$ to $[0, 2^{25} - 1]$
- μ: from $[0, 20]$ to $[0, 2^{25} - 1]$
- σ: from $[0, 1000]$ to $[0, 2^{17} - 1]$
- δ: from $[0, 25000]$ to $[0, 2^{15} - 1]$

The Andouin's population data at La Banya from 2006 to 2017.

	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017
Population	15329	14177	13031	9762	11271	8688	7571	6983	4778	2077	1580	793

Chapter 2

Programme Structure

The programme is structured in three distinct parts.

- On the one hand, the initial part in which we randomly create the initial values of each chromosome of our population.
- On the other hand, the part in which we calculate the fitness of each chromosome, and rearrange them in such a way that the chromosome with the lowest fitness occupies the first place in our array of structures.
- Finally, the part where we generate a new population.

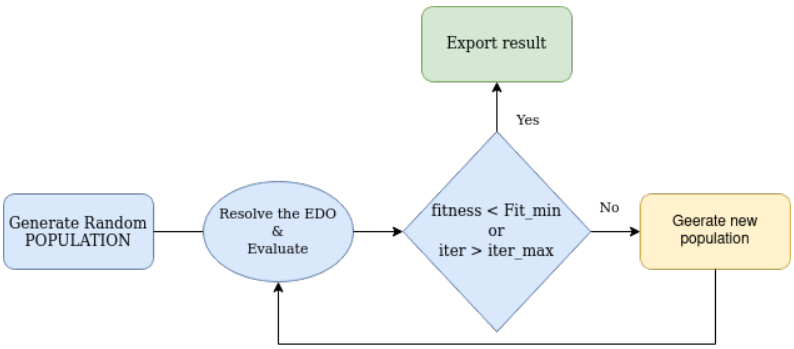


Figure 2.1: General structure of the algorithm.

For the execution of the programme, in total, we have made use of two different structures. On the one hand, the first structure takes care of storing all the data about the chromosomes:

```
typedef struct{
    unsigned long int x0;
    unsigned long int phi;
    unsigned long int lambda;
    unsigned long int mu;
    unsigned long int sigma;
```

```
    unsigned long int delta;
    double beta;
    double predicted_values[nYears];
    double fitness; // the fitness of the chromosome
    double p;
    double q;
}Type_chromosome;
```

As we can see here, all the parameter are stored as an unsigned int with the exception of β. On the other hand, once we run the function to solve the ODE, we obtain a result for each year. So, we have created a vector of 12 double parameters called *predicted_values*. Finally, we store three parameters associated with the precision of our results: a double with the value of the fitness and then the value of p and q (values used only in the roulette selection).

Additionally, we have created an structure called ODE_Parameters that will simply contain the values of the six parameters in double format. Thus each time we have a new population of chromosomes, we have to fill this structure (using the values within the principal structure and multiplying them by each conversion factor) in order to resolve the ODE and create new results.

```
typedef struct {
    double x0;
    double phi;
    double beta;
    double lambda;
    double mu;
    double sigma;
    double delta;
} ODE_Parameters;
```

Once the two main structures have been created, we initialise the program by giving random values to each of the parameters of the structure *Type_chromosome*. Then, we make the conversion to decimal numbers and solve the ODE using the Runge-Kutta-Fehlberg method of order 7-8 with adaptive space.

Subsequently, we use the results of the ODE to compute the fitness of each chromosome using the 1.2 equation. Using the fitness we can now sort the chromosomes in ascending order. In this way we will have more accessible the best results of each population.

Now, we have two possible strategies:

- On the one hand, we can generate a random number and, as long as this number is lower than a certain parameter (chosen by us), we will allow the reproduction of two chromosomes. this
- On the other hand, and this is the path we will follow, we will reserve the 10% of the population with the best results for the next generation and we will use the remaining 90% for reproduction (crossover). In this way we make sure that the best individuals continue into the next generation.

The two chromosomes chosen for crossover will be selected by the *tournament selection*:

```
int tournament_selection_2(Type_chromosome chromosome[], unsigned int t, unsigned
    int N){
    int i, j, best_index, max_random;
    unsigned int index;
    double best_fitnes = DBL_MAX, first_election;

    first_election = uniform();

    if (first_election < 0.80){
        max_random = N * 0.2;
        for (j = 0; j < t; j++){
            index = uniform() * (max_random - 1);
            if (chromosome[index].fitness < best_fitnes){
                best_index = index;
                best_fitnes = chromosome[index].fitness;
            }
        }
        return best_index;
    }

    else{
        max_random = (0.2 + uniform() * 0.8) * N;
        for (j = 0; j < t; j++){
            index = uniform() * (max_random - 1);
            if (chromosome[index].fitness < best_fitnes){
                best_index = index;
                best_fitnes = chromosome[index].fitness;
            }
        }
        return best_index;
    }
}
```

As we can see in the previous code snippet, we assign a certain probability of selection depending on whether the individual's fitness is lower or higher. That is, to the first 20% of the population (remember that we have ordered the individuals in ascending order), we assign a probability of 80% of being selected. To the remaining 80% however, we give a probability of 20%. In this way we favour the reproduction among the best-fit individuals.

Once we have selected the two individuals to reproduce, we must make a decision on what we want our algorithm to look like. This decision will be represented by a constant called *keep*.

If we want our genetic algorithm to be more exploratory, we will reserve a smaller number of chromosomes for the next generation and reproduce a larger number of them. In this case *keep* takes a larger value (always between $[1, POP\_SIZE]$).

Conversely, if we decide to make our algorithm more exploitative, we will reserve a larger number of individuals (those with the lowest fitness) and leave a small part of the remaining individuals for crossover.

Finally, in order to maintain genetic diversity from the old generation to the next we mutate all of the individuals.

After all this process, we evaluate the new individuals once more and we decide whether to continue with a new iteration (repeat the whole process once again).

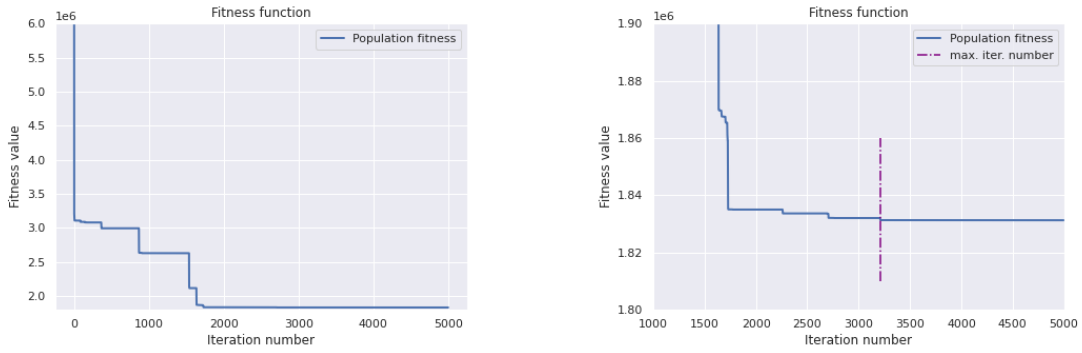
Chapter 3

Results

In this section we will present the results obtained in the work.

As explained in the previous section, the results will vary depending on how exploratory or exploitative we want our algorithm to be. In our case, the value of *keep* that has given us the best results is 8. In other words, in each iteration we have kept the best $POP\_SIZE/8$ chromosomes for the next population while we have carried out the crossover with the rest of the members.

Therefore, using *keep*= 8, a population size of 100 individuals and 5000 iterations, the evolution of our fitness has behaved in the following way:



(a) (b)

Figure 3.1: total fitness evolution (a) and partial fitness evolution (b) vs number of iteration. The vertical purple line indicate the final convergence value.

Thus, using the values motioned before we have obtained a a minimal fitness value of 1831237.471306.

On the other hand, the final values for x(t) compared to real values are as follows:

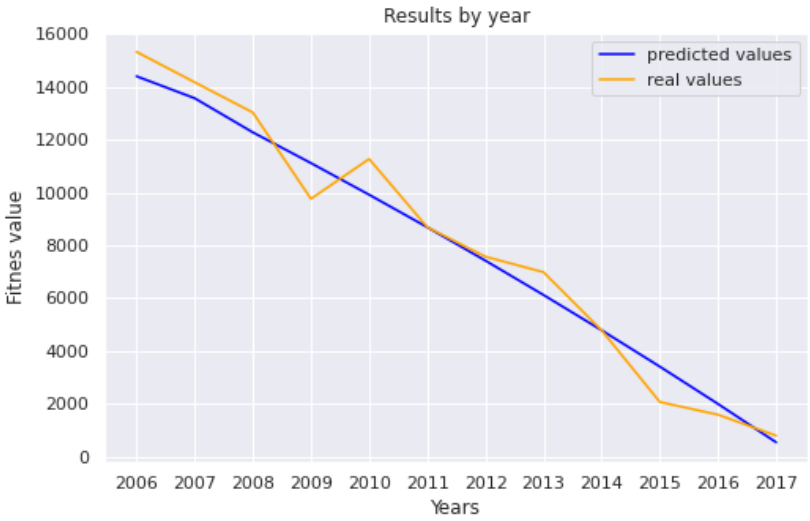


Figure 3.2: Differences between real and predicted results of x(t) for the time period 2006-2017.

We have also saved the results obtained in each iteration, both the values of the fitness and the values of x(t) as well as the values of each parameter of the equation in the following files:

- fitness_results.txt: here you can find the fitness of each iteration form 0 to 5000.
- params_results.txt: here you can find all the parameter values for each iteration.
- params_results.txt: here you can find the result of the ODE for each epoch from 1 to 5000.

All of the code is available in the following GitHub repository.

Chapter 4

Conclusions

In this work we have tested the hypothesis that Andouine's migration occurs with social copying using a genetic algorithm. The genetic algorithm has been designed for the minimisation of the fitness value defined in the first section and has given us a minimum result of 1.8M. Although we cannot affirm that the algorithm has converged to values very close to the real solution, we can conclude that, taking into account the large search space of each parameter, the results have been acceptable.

The algorithm has been developed according to the following steps. In the first part, we have defined two different structures necessary for the proper functioning of the programme. On the one hand, we have defined a main structure composed of the parameters in the form of unsigned int. On the other hand, we have built a parallel structure with all the values of the parameters as doubles. This last structure has been created simply to create solutions to the ODE presented in the first section.

In the second part we have made use of the previously created structures to generate a population of structures (or chromosomes) at random. In addition, each individual in the population has been assigned a *fitness* value depending on how well the parameters have been matched to the actual result. Next, we selected a certain number of individuals for the reproduction. Specifically, we reserved one eighth of the individuals with the best fitness for the next population and selected the remaining pairs of individuals based on the tournament selection.

Within the selection of pairs of individuals for breeding we have favoured with an 80% probability the first set of chromosomes with the best fitness. Furthermore, we have set the value of random individuals with a chance of being selected to $POP\_SIZE * 0.9$.

Another method used for the selection of individuals (of parents) was roulette selection. For this we used the variables p, and q. However, the results obtained for this method were not at all conclusive and we could not get the algorithm to converge. We have therefore decided to leave it aside and continue with the method explained in the previous paragraph.

Finally, once the pairs of individuals had been selected, we proceeded to carry out both the cross over and the mutation. The latter has been applied to the whole population.

Once this process has been completed, we have repeated it again and again until approximately at iteration number 3210 the algorithm has reached its minimum value and has stabilised.